NASA CR. 143993

FORTRAN AUTOMATED CODE EVALUATION SYSTEM (FACES)

FUNCTIONAL CHARACTERISTICS

Version 2, Mod 0

September, 1975

Contract : NAS8-30928

(NASA-CR-143993) FORTRAN AUTOMATED CODE EVALUATION SYSTEM (FACES) FUNCTIONAL CHARACTERISTICS, VERSION 2, MOD 0 (Brown and Ramamoorthy, Inc., Berkeley, Calif.) 75 p HC $4.25 CSCL 09B G3/61

N76-10754

Unclas 42347

BROWNE & RAMAMOORTHY, INC.

2550 Telegraph Avenue, Suite 404, Berkeley, Ca., 94704

(415) 848-0261

Table of Contents

I. Introduction

Functional Characteristics are a level of description too detailed for a User's Manual, yet too abstract for detailed documentation. This information describes the external behavior of major subsystems or the total system. Some descriptions are simply summaries of information available in scattered locations.

Codes and capabilities are likely to change with system modification and extension. This material is provided as a detailed reference source for personnel familiar with the FACES system.

II. DETAILED DESCRIPTION OF FACES ACCEPTABLE FORTRAN

FACES is designed to operate with a compiler. The compiler is responsible for policing acceptable constructions which can be executed; FACES is responsible for analyzing syntax which is compiler acceptable.

In defining the syntax for FACES, maximum latitude is given to form. Where possible, features are a composite of nonconflicting capabilities from several compilers. The presence of a particular construction implies the source code can be compiled.

The following description is intended to define syntactic constructions recognized by FACES; this is not a description of FORTRAN. A FORTRAN user's guide should be consulted for an explaination of code operation.

Character Set

	Blank character (significant only in Hollerith literal strings)
A-Z	Alphabetic characters
0-9	Decimal numeric characters

Special Characters

=	equal sign	+	plus sign
-	minus sign	*	asterisk
/	slash	,	comma
(	left parenthesis	)	right parenthesis
.	period	$	dollar sign
'	apostrophe (single quote)		
"	quotation mark (double quote)		

Card Format

Standard FORTRAN format with numeric statement labels in columns 1 through 5, continuation in column 6, and source statement text in columns 7 through 72. Columns 73 through 80 may contain optional card identifications.

Statement labels.

Statement labels are composed of from 1 to 5 unsigned numeric characters without leading zeroes.

Continuation cards.

Statements may be continued on the next card by placing any character other than blank or zero in column 6.

Comment Cards

Cards containing any character other than blank or a numeric character in column 1 are considered comment cards. Blank cards are considered comment cards.

Source Code Components

Symbolic Names. Symbolic names are from 1 to 8 alphanumeric characters, the first of which must be alphabetic.

Constants.

1. Integer Constants. A string of decimal digits not formally constrained in length.
2. Real Constants. Real constants have one of the following forms:

   i.i .i i. basic real constants

   <basic real constant> E j

   i E j

   where: i and j are integer constants.

   The basic real constant and exponent may be optionally signed.
3. Double Precision Real Constants. Double precision constants have the form,

   <basic real constant> D j

   where: the basic real constant and integer exponent may be optionally signed.
4. Complex Constants. The form of complex constants is,

   (c1 , c2)

   where: c1 and c2 are either real or double precision constants. Mixture of precision is permitted. c1 and/or c2 may be signed.
5. Logical Constants. Logical constants may be one of the following forms,

   .TRUE. .FALSE. .T. .F.

6. Literal Constants. Literal constants are character data of the form,

   w H ⟨char string⟩

   ' ⟨char string⟩'

   " ⟨char string⟩"

   where: ⟨char string⟩ is a series of adjacent card characters including blanks.

   w is an unsigned integer constant indicating the length of the character string.

   If two adjacent delimiting markers (i.e. '' or "") appear they are interpreted as a single mark of text.

7. Nondecimal Based Constants. Machine dependent constant forms for radix other than 10.

   w Z ⟨hex constant string⟩

   0 ⟨octal constant string⟩ (more than 7 chars)

   ⟨octal constant string⟩ B

   Z ⟨hex constant string⟩ (in DATA list only)

Operators.

1. Arithmetic operators. The following arithmetic operators are permitted:

   + - / * **

2. Logical operators. The following logical operators are permitted:

   .NOT. .N. .AND. .A. .OR. .O.

3. Relational operators. The following relational operators are recognized:

.EQ. .NE. .GT. .GE. .LE. .LT.

Program variables.

Variable names are limited to 8 or less alphanumeric characters, the first of which must be alphabetic.

Array dimensions are unlimited. Dimension specification must be declared by either an unsigned constant or unsigned variable. Array reference subscripts may be any arithmetic expression.

Data Types. The following types are processed by FACES:

INTEGER

LOGICAL

REAL

DOUBLE PRECISION

COMPLEX

HOLLERITH (only as a constant)

NEUTRAL (untyped routine names, statement labels, and COMMON block labels)

Expressions. Expressions are processed by content only. Expression syntax is not examined. Operator precedence is ignored. Any valid expression acceptable to a FORTRAN compiler will be accepted by FACES if components of the expression contain acceptable operators and operands.

1. Simple Arithmetic Expressions.

Simple arithmetic expressions contain the following components:

Operands: Constants, scalar variables, array references without subscript lists.

Operators: Arithmetic operators, and logical operators.

Other: Organizing parentheses.

2. Arithmetic Expressions: Arithmetic expressions contain the the following components:

   Operands: Constants, scalar variables, arrays with unrestricted subscripts, function references, and statement function references.

   Operators: Arithmetic operators and logical operators.

   Other: Organizing parentheses.

3. Logical Expressions: Logical expressions contain the following components:

   Operands: Constants, scalar variables, arrays with unrestricted subscripts, function references, and statement function references.

   Operators: Arithmetic operators, logical operators, and relational operators.

Expression type is evaluated only for expressions used as array subscript references, or actual parameters to functions or subroutines. The type is determined by the highest level type of any operand in the expression:

COMPLEX

DOUBLE PRECISION

REAL

INTEGER

LOGICAL / HOLLERITH (extended type)

Nesting of function and array references within function actual parameters, array subscripts, and subroutine actual parameters is currently limited to 5 levels.

SOURCE CODE STATEMENTS

Control Statements. Branch targets, t, in control statements may be either statement labels or variables set by ASSIGN statements. A branch target list is a series of branch targets separated by commas.

GO TO Statements.

GO TO t (unconditional GO TO)

GO TO (⟨branch target list⟩), v

ASSIGN & TO v

GO TO v, (⟨branch target list⟩) (ASSIGNED GO TO)

where t is a branch target

v is and unsubscripted variable

& is a statement label

The branch target list of the ASSIGNED GO TO is optional

IF Statements.

IF(⟨arith expr⟩) t1, t2, t3

IF(⟨logical expr⟩) t1, t2

IF(⟨logical expr⟩) statement

where t1, t2, and t3 are branch targets

⟨arith expr⟩ is an arithmetic expression

⟨logical expr⟩ is a logical expression.

DO & ⟨DO control list⟩

where & is a statement label.

⟨DO control list⟩ is the form,

v = n1, n2, n3

with v and unsubscripted variable.

n1, n2, n3, the optionally signed DO control parameters.

n1, n2, n3, may be either constants or unsubscripted variables.

n3 may be omitted.

CONTINUE

Identified control statements.

The following control statements may optionally contain an identification, n, indicated by an unsigned constant or symbolic name.

PAUSE n

STOP n

END n

Assignment Statements. Assignment statements have the usual form,

v = ⟨arith expr⟩

where v is a subscripted or subcripted variable.

⟨arith expr⟩ is an arithmetic expression.

Input/Output Statements. I/O statements are composed of an I/O action, I/O control specification, and optional I/O variable list.

I/O control specifications are of the form,

(u'r, f, ERR = b1, END = b2)

where u is a mandatory unit specification expressed as a constant or unsubscripted variable.

r is an optional record specification restricted to a simple constant or unsubscripted variable.

f is an optional FORMAT specification expressed as a statement label or variable name.

b1 and b2 are branch targets.

I/O lists are comma separated lists of I/O elements.

I/O elements are:

1. Simple operands (constants or variables)
2. I/O elements separated by commas enclosed in parentheses.
3. Implied DO loops of the form

   (⟨I/O element list⟩, ⟨DO control list⟩)

I/O Statement Forms.

WRITE ⟨I/O Control Spec⟩ ⟨I/O list⟩

READ ⟨I/O control spec⟩ ⟨I/O list⟩

PRINT f, ⟨I/O list⟩

PUNCH f, ⟨I/O list⟩

where f is a format specification

the I/O list is optional

END FILE u

REWIND u

BACKSPACE u

where u is an I/O unit specification

FORMAT only the statement label of a FORMAT is processed.

Variable Declaration Statements. Variable declarations define the data type, structure, and storage allocation of program variables.

Type specifications may be one of the following

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

LOGICAL

Unless redefined by an IMPLICIT statement, the leading character of the variable establishes ANSI standard type.

A through H and O through Z - type REAL

I through N - type INTEGER

A variable declaration list is a comma separated list of variables which may be optionally subscripted. Subscript specification implies difinition of array dimensions.

A variable list is a comma separated list of subscripted or unsubscripted variable which reference program variables or elements of arrays.

IMPLICIT ⟨type spec⟩(⟨letter list⟩), ... ,
⟨type spec⟩(⟨letter list⟩)

where ⟨type spec⟩ is a defined type specification.

⟨letter list⟩ is a comma separated form with elements of:

1. Single letter
2. Range of letter specified as L1-L2.

   L2 must be alphabetically ahead of L1.

DIMENSION ⟨var declr list⟩

⟨type spec⟩⟨var declr list⟩

EQUIVALENCE (⟨var list⟩),(⟨var list⟩), ... ,(⟨var list⟩)

DATA ⟨var list⟩/⟨data spec list⟩/. ... ,

⟨var list⟩/⟨data spec list⟩/

where ⟨data spec list⟩ is a comma separated list with elements of:

1. Optionally signed simple constants
2. p * c where p is an unsigned repeat specification and c is an optionally signed constant.

COMMON ⟨block spec⟩ ⟨var declr list⟩ ....

⟨block spec⟩ ⟨var declr list⟩

where ⟨block spec⟩ is the form / symbolic name /

If the first ⟨block spec⟩ is absent, the COMMON Block is blank COMMON.

A missing symbolic name explicitly references blank COMMON (e.g. / /).

Subprocess Statements.

Dummy parameter lists are comma separated lists of unsubscripted variables enclosed in parentheses.

Actual parameter lists are comma separated lists of arithmetic expressions, the simplest form of which is a variable or constant, enclosed in parentheses.

PROGRAM name

where name is the symbolic name of the main program.
Optional text following name is not processed.

⟨type spec⟩ FUNCTION name ⟨dummy parm list⟩

where ⟨type spec⟩ is optional

SUBROUTINE name ⟨dummy param list⟩

where ⟨dummy param list⟩ is optional

BLOCK DATA

ENTRY name ⟨dummy param list⟩

where name is the symbolic name of a secondary entry point.
The type for name is derived independently of the primary entry.
⟨dummy param list⟩ is optional.

sfname ⟨dummy param list⟩ = ⟨arith expr⟩

where sfname is the symbolic name of a statement function distinct from any declared array.

EXTERNAL name1, name2, ..., nameN

where nameI is the symbolic name of an external function or subroutine.

RETURN i

where i is an optional constant or unsubscripted variable return specification.

Function reference

name ⟨actual param list⟩

where name is the symbolic name of an external function or function entry point.

CALL name ⟨actual param list⟩

where name is the symbolic name of a subroutine.

⟨actual param list⟩ is optional.

Statement Order Requirements.

Minimum order requirements are implemented in FACES to provide maximum flexibility to FORTRAN dialects consistent with single pass operation.

1. All modules must begin with a header card identifying the module. Header cards are PROGRAM, BLOCK DATA, SUBROUTINE, and FUNCTION.
2. All modules must terminate with an END card.
3. References to arrays must appear after declaration of the array in a DIMENSION, TYPE, or COMMON statement to avoid ambiguity with statement function definitions and function references.

Other Requirements.

Since modules are identified and replaced by name, only one BLOCK DATA module is permitted in the software system.

Transition Recording Codes for TRANS

Parameter Value	Type of Transfer	Values Recorded PPCode	Predecessor	Successor
1	Branch to label	1	Node number of branch statement	Symbol table position of branch label
2	DO LOOP	2	Symbol table position of DO label	Node number of DO statement
3	RETURN	0	Node number of RETURN	Special Return code (20000)
4	Logical IF statement	0	Node number of IF statement	Node number of statement after IF
		0	Node number of IF statement	Node number of second statement after IF
5	External procedure reference	0	Node number of calling statement	Special Call Code (10000)
6	END	0	Node number of END statement	Special END code (30000)
7	Program termination	0	Node number of terminating statement	Special transition code (40000)
8	Primary entry point	0	Special code for primary entry (70000)	Node number of module header card
9	Secondary entry point	0	Special code for secondary entry (80000)	Node number of ENTRY statement
10	Branch through variable	0	Node number of branch statement	Special code for variable branch (60000)
11	Not used			
12	Statement function reference	0	Node number of statement calling statement function	Special code for statement function reference (50000)

```
FORTRAN FRONT END CODE SUMMARY
DIRECTORY CODES
    MODULE TYPE
    0-UNDEF     1-PROG     2-SUB      3-FUNC     4-BLKDATA
    5-SECND SUB ENTRY      6-SECND FUN ENTRY

SYMBOL CODES
    CHARACTERISTICS
    TYPE CODES -
     0-UNDEF  1-FP  2-DP  3-CMPX  4-LOG  5-NTRL  6-CHAR   7-INTG
    CLASS CODES -
     0-INDEF     1-SUP     2-STFUN     3-ARRAY     4-FUN        5-LABEL
     6-SCALAR    7-COMLAB  8-CONS      9-ENTRY     10-          11-
    12-PROG      13-TEMP   14-         15-STF DUM  16-EXTPROC

USE TABLE CODES
    USE CODES
     0- EMPTY          1- ASGN OVAR    2- ASGN IVAR    3- I/O OVAR
     4- I/O INVAR      5- DO INDX VAR  6- DO START     7- DO END
     8- DO INC         9- LAB DEF      10- XFR TO LAB  11- COM ENTRY
    12- DATA ENTRY     13- ARRY DECLR  14- TYP ENTRY   15- SUBSCRIPT
    16- FUN DUMMY      17- SUB DUMMY   18- FUN ACTUAL  19- SUB ACTUL
    20- CONDBR VAR     21- EXT PROC    22- ASSIGN VAR  23- BR THRU VAR
    24- CGOTO INDX     25- DO LABEL    26- I/O UNIT    27- FORMAT REF
    28- MTPL RETRN     29-REF LAB LOC  30- BGN LIST    31- END LIST
    32- BGN SBE        33- END SBE     34- SBE OVAR    35- SBE IVAR
    36- EQUIV ENTY     37-EXTRN ENTY   38-             39-
    40- DECLARE        41- DATA VAL    42- REPEAT      43- ID INDEX
    44-                45-             46-ST FUN CALL  47- I/O RECRD

NODE TABLE CODES
    STATEMENT TYPES -
     0- UNDEF          1-              2-              3- FORMAT
     4- PRINT          5- IMPLICIT     6- NAMELIST     7- ENCODE
     8- DECODE         9- PUNCH        10- IF          11- COMPLEX
    12- EXTERNAL       13- BLOCK DATA  14- END         15- READ
    16-                17-             18-             19-
    20- ENDFILE        21-             22-             23- REAL
    24- BACKSPACE      25-             26- LOGICAL     27- FUNCTION
    28- DIMENSION      29-             30- SUBROUTINE  31- DATA
    32- PROGRAM        33- COMPL REF   34- CALL        35- INTEGER
    36- COMMON         37-             38- ASSIGN      39- WRITE
    40- EQUIVAL        41- STOP        42- REWIND      43-
    44- CONTINUE       45- GOTO        46-             47-
    48- ENTRY          50- PAUSE       51- RETURN      52- ASGNMT STMT
    53- DO STMT        54- ST FUN
    99- UNRECOGNIZED

SUCCESSOR TABLE
    SUCCESSOR CODES
    0 - UNDEF          1 - 9999    STATEMENT NUMBER
    10000 - EXT REF      20000 - RETURN       30000 - END STMT
    40000 - PROG HALT    50000 - STFUN REF    60000 - BR THRU VAR
    90000 - UNDEF LABEL

PREDECESSOR TABLE

    PREDECESSOR CODES
    0 - UNDEF          1 - 9999    STATEMENT NUMBER
    70000-PRIME ENTRY    80000-SECND ENTRY
    90000-UNDEF LABEL
```

Summary of System Anomalies Detected by FORTRAN Front End

FORTRAN Front End anomalies are detected processing conditions which indicate unusual circumstances. Anomalies should not appear if the FORTRAN text is well formed and tables do not overflow. Unusual syntax or table overflow may cause anomalies to appear although the error should be controlled.

Anomalies are reported in the form,

***** FFE SYSTEM ANOMALY # DETECTED BY RRRRRRRR IN
NNNNNNNN CARD CCC VALUES = III AAAAAAAA

where,

is an anomaly code indicating the type of situation which occurred,

RRRRRRRR is the FFE routine which detected the error,

NNNNNNNN is the module being processed when the anomaly was detected,

CCC is the (approximate) card number relative to the beginning of the module,

III is the value found in question, and

AAAAAAAA is the name of the data element.

Table of FFE Anomaly Code

Code Number	Meaning
1	Bad parameter value passed
2	Bad table value detected
3	Bad table positioning detected
4	Unexpected sequence encountered
5	Probable lockup cleared
6	Unacceptable form found
7	Incapable of performing the requested function
8	Algorithm produced suspicious result
9	Processing table overflow

QUERIES

FACES allows a user to request the search for certain pre-determined unusual language constructions in the user's FORTRAN source code. Such a search is called a query. The user also has some control over the format of the messages produced when such incongruous constructions are located.

110 (ANSIST) - Search for ANSI Standards function names that are not being used as ANSI Standards functions names. A list of these names appears in Table 2, which was derived from the ANSI FORTRAN manual ANSI X3.9-1966. All messages appear in the primary listing.

Example: DIMENSION IFIX(10)
ABS = IFIX(MOD) (ABS and MOD are variables)

120 (RESWRD) - Search for FORTRAN 'Reserved' words being used as names. A list of these words appears in Table 1. All messages appear in the primary listing.

Example: DIMENSION IF(5,5)
DO = IF(4,5) (DO and IF are variables)

130 (DATVAR) - Search for DATA statements not in BLOCK DATA which contain COMMON Block variables. These DATA satements are loader dependent. All messages appear in the primary listing.

ASSIGN	DO	GOTO	READ
CALL	ENCODE	IF	REAL
COMMON	END	IMPLICIT	RETURN
COMPLEX	ENTRY	LOGICAL	REWIND
CONTINUE	EXIT	PAUSE	STOP
DATA	FORMAT	PRINT	WRITE
DECODE	FUNCTION	PUNCH	

"Reserved" FORTRAN Words

Table 1

ABS	ATAN2	DATAN	DSIN	MAX1
AINT	CABS	DATAN2	DSQRT	MINO
ALOG	CCOS	DBLE	EXP	MIN1
ALOG10	CEXP	DCOS	FLOAT	MOD
AIMAG	CLOG	DEXP	IABS	REAL
AMAXO	CMPLX	DIM	IDIM	SNGL
AMAX1	CONJG	DLOG	IDINT	SIGN
AMINO	COS	DLOG10	IFIX	SIN
AMIN1	CSIN	DMAX1	INT	SQRT
AMOD	CSQRT	DMOD	ISIGN	TANH
ATAN	DABS	DSIGN	MAXO	

Standard FORTRAN Function Names

Table 2

If the COMMON Block variable is EQUIVALENCED to a variable which appears in a DATA statement, then both variable names appear in the message.

```
Example:  SUBROUTINE SUB
          COMMON X,Y,Z
          EQUIVALENCE (Y,Q)
          DATA X,Q/7.3, 9.5/          (Set values of X and Y)
```

140 (FUNPAR) - Search for function dummy parameters assigned values within the function itself. These represent dangerous side effects. All messages appear in the primary listing.

If a dummy parameter is EQUIVALENCED to a local variable which is assigned a value, then both variable names appear in the message.

Program boundaries are not crossed. The use of a dummy parameter in an actual parameter list is ignored for this query.

```
Example:  FUNCTION FUN(X,Y)
          EQUIVALENCE (Y,Q)
          X = X+3                     (X value changed)
          Q = X                       (Y value changed)
```

150 (MULBRA) - Search for multiple branching statements which do not branch to the statement immediately following. These are highly questionable logical constructions. All messages appear in the primary listing.

Example: IF (K) 100, 200, 300

50 C = FUN(I)

160 (REDLOP) - Search for the redefinition of DO Loop control variables within the loop itself. Such redifinitions are illegal. All messages appear in the primary listing.

If the control variable is EQUIVALENCED to some other variable which is assigned a value within the loop, both variable names appear in the message.

Program boundaries are not crossed. If a DO Loop control variable appears in an external reference as a parameter, it is assumed that the value of the control variable is not changed in the external reference.

```
Example:  EQUIVALENCE (J,M)
          DO 100 I = J,K,L
             M = I              (modifies J)
             I = I+1            (modifies I)
 100      CONTINUE
```

170 (DOTERM) - Search for DO Loop index variables used after the DO Loop has terminated normally. For many compilers, the DO Loop index variable is undefined after the loop terminates under normal conditions. All messages appear in the primary listing.

If the DO Loop index variable is EQUIVALENCED to a local variable which is used after the loop terminated normally, then both variable names appear in the message.

Program boundaries are not crossed. If the index variable appears in a parameter list, it is assumed that the index variable is not used as an input parameter.

171 (DOTERM) - Same a 170, except that the messages appear in the secondary listing.

The message in the secondary listing contains the following:

1. The first statement in the module.
2. The statement containing the beginning of the DO Loop.
3. The statement containing the end of the DO Loop.
4. All statements in the path(s) leading from the end of the DO Loop to the use of index variable.
5. The name of the DO Loop index variable.
6. If the DO Loop index variable is EQUIVALENCED to a local variable which is used after the loop terminated normally, the equivalenced name appears in the secondary listing.

Example:

```
      EQUIVALENCE (J,K)
      I = 0
   10 I = I+1                  (use I on backward branch)
      DO 100 I = 1,10
         DO 100 J = 1,5
  100 CONTINUE
      LENGTH = K               (use of J)
      GO TO 10
```

180 (ASNUSE) - Search for local variables assigned values but never used. These variables often represent keypunch errors or historical legacies. All messages appear in the primary listing.

```
Example:  SUBROUTINE SUB(B)
          P = FUNC(B)                (P assigned but not used)
          RETURN
          END
```

190 (UNINT) - Search for uninitialized local variables. All messages appear in the primary listing.

Program boundaries are not crossed. If the variable appears in a parameter list, it is assumed that the variable receives a value within that external reference.

191 (UNINT) - Same as 190, except that the messages appear in the secondary listing.

The message in the secondary listing contains the following:

1. The first statement in the module.
2. All the statements in the path(s) leading from the first executable statement in the module to the use of the uninitialized variable.
3. The name of the uninitialized variable.

```
Example:  SUBROUTINE SUB(A,B )
          A = B+C                    (C is uninitialized)
          RETURN
          END
```

400 (CBNENT) - Search for corresponding COMMON Block declarations which do not have the same number of entries. Such constructions are highly error-prone and some are machine dependent. (see example). All messages appear in the primary listing.

Each message contains:

1. The name of the COMMON Block.
2. The number of entries in the COMMON Block.
3. The name of module the corresponding COMMON Block declaration appears in.

401 (CBNENT) - Same as 400, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and for the comparison COMMON Block declaration:

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The number of entries there are in the COMMON Block.

Example:

```
SUBROUTINE SUB1
COMMON A,B,C
COMMON/COM/X,Y,Z

SUBROUTINE SUB2
COMMON D(3)
COMMON/COM/X,Y
```

410 (CBTYPE) - Search for corresponding entries in corresponding COMMON Block declarations which do not have the same type. Such constructions are highly error-prone. The comparison of two declarations halts after the first mismatch is located.

Each message contains:

1. The name of the COMMON Block.
2. The name of the COMMON Block entry.
3. The entry's type.
4. The entry number. This indicates where the entry appears in the COMMON Block declaration.
5. The name of the module the corresponding COMMON Block declaration appears in.

411 (CBTYPE) - Same as 410, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and for the comparison COMMON Block declaration,

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The name of the COMMON Block entry.
5. The entry's type.
6. The entry number. This indicates where the entry appears in the COMMON Block declaration.

Example:

```
SUBROUTINE SUB1
COMMON A,B,C

SUBROUTINE SUB2
COMMON A,B,I
```

420 (CBDIM) - Search for corresponding entries in corresponding COMMON Block declarations which do not have matching dimension. Such constructions are highly error-prone. The comparison of two declarations halts after the first mismatch is located. All messages appear in the primary listing.

Each message contains:

1. The name of the COMMON Block
2. The name of the COMMON Block entry.
3. The number of dimensions the entry has, from zero to n.
4. The entry number. This indicates where the entry appears in the COMMON Block declaration.
5. If the entry is an array, the actual dimensions of the entry.
6. The name of the module the corresponding COMMON Block declaration appears in.

421 (CBDIM) - Same as 420, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and for the comparison COMMON Block declaration,

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The name of the COMMON Block entry.
5. The number of dimensions the entry has, from zero to n.
6. The entry number. This indicates where the entry appears in the COMMON Block declaration.
7. If the entry is an array, the actual dimensions of the array.

```
Example:  SUBROUTINE SUB1
          COMMON A(1), B(2)
          COMMON/COM1/C(10),D
          COMMON/COM2/E(2,3)

          SUBROUTINE SUB2
          COMMON A,B
          COMMON/COM1/C(11)
          COMMON/COM2/E(3,2)
          DIMENSION B(2)
```

430 (CBONE) - Search for COMMON Blocks which appear only once in a software system. This often represents a keypunch error or a historical legacy. All messages appear in the primary listing.

440 (CBNAME) - Search for corresponding entries in corresponding COMMON Block declarations which do not have the same name. Such constructions are confusing and error-prone. The comparison of the declarations halts after the first mismatch is located. All messages appear in the primary listing.

Each message contains

1. The name of the COMMON Block.
2. The name of the COMMON Block entry.
3. The entry number. This indicates where the entry appears in the COMMON Block declaration.
4. The name of the module the corresponding COMMON Block declaration appears in.

441 (CBNAME) - Same as 440, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and for the comparison COMMON Block declaration,

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The name of the COMMON Block entry.
5. The entry number. This indicates where the entry appears in the COMMON Block declaration.

Example:
```
SUBROUTINE SUB1
COMMON A,B,C
COMMON/COM1/X,Y

SUBROUTINE SUB2
COMMON A,C,B            (B and C transposed)
COMMON/COM1/X(2)
```

450 (CBINDS) - Search for corresponding entries in corresponding COMMON Block declarations which do not have the same individual size. Such constructions are error-prone. The comparison of two declarations halts after the first mismatch is located. All messages appear in the primary listing.

Each message contains:

1. The name of the COMMON Block.
2. The name of the COMMON Block entry.
3. The size of the entry, in computer words.

4. The entry number. This indicates where the entry appears in the COMMON Block declaration.
5. The name of the module the corresponding COMMON Block declaration appears in.

451 (CBINDS) - Same as 450, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and the comparison COMMON Block declaration,

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The name of the COMMON Block entry.
5. The size of the entry, in computer words.
6. The entry number. This indicates where the entry appears in the COMMON Block declaration.

Example:
```
SUBROUTINE SUB1
COMMON A(3),B(3)
COMMON/COM1/DP
DOUBLE PRECISION DP

SUBROUTINE SUB2
COMMON A(4),B(2)
COMMON/COM1/DP
INTEGER DP
```

460 (CBTOTS) - Search for corresponding COMMON Block declarations which do not have the same total size. Such constructions are highly error-prone and are machine dependent. All messages appear in the primary listing.

Each message contains

1. The name of the COMMON Block.
2. The total size of the COMMON Block, in computer words.
3. The name of the module the corresponding COMMON Block declaration appears in.

461 (CBTOTS) - Same as 460, except that the messages appear in the secondary listing and that each message contains, for both the model COMMON Block declaration and the comparison COMMON Block declaration,

1. The first statement in the module.
2. The statement containing the COMMON Block declaration.
3. The name of the COMMON Block.
4. The total size of the COMMON Block, in computer words.

Example:

```
SUBROUTINE SUB1
COMMON A,B,C
COMMON/COM1/DP
DOUBLE PRECISION DP

SUBROUTINE SUB2
COMMON A,B,C,D              (longer by 1 variable)
COMMON/COM1/DP              (size difference caused by
                             storage allocation)
```

500 (PLNENT) - Search for corresponding parameter lists which do not have the same number of entries (parameters). Such constructions are highly error-prone and are machine dependent. The model for comparison is always the formal (dummy) parameter list of a subprogram. The name of an external reference (a subprogram) is considered to be the first entry in its own parameter list. All messages appear in the primary listing.

Each message contains

1. The number of parameters in the parameter list.
2. The name of the module which contains the corresponding parameter list.

501 (PLNENT) - Same a 500, except that the messages appear in the secondary listing and that each message contains, for both the formal parameter list and for the actual parameter list,

1. The first statement in the module.
2. The statement containing the parameter list.
3. The number of parameters in the parameter list.

Example:
```
CALL SUB1
K = IX(M,N)

SUBROUTINE SUB1(A)

FUNCTION IX(M)
```

510 (PLTYPE) - Search for corresponding entries (parameters) in corresponding parameter lists which do not have the same type. Such constructions are highly error-prone. The model for comparison is always the formal (dummy) parameter list of a subprogram. The name of an external reference (a subprogram) is considered to be a parameter.

All messages appear in the primary listing.

Each message contains

1. The name of the parameter.
2. The parameter's type.
3. The parameter number. This indicates where the parameter appears in the parameter list.
4. The name of the module which contains the corresponding parameter list.

511 (PLTYPE) - Same as 510, except that the messages appear in the secondary listing and that each message contains, for both the formal parameter list and for the actual parameter list,

1. The first statement in the module.
2. The statement containg the parameter list.
3. The name of the parameter.
4. The parameter's type.
5. The parameter number. This indicates where the parameter appears in the parameter list.

Example:
```
Q = FUNC(X,Y,Z)                          (reference to REAL FUNCTION)
CALL FUNC(X,Y,Z)          (SUBROUTINE reference to a FUNCTION)

INTEGER FUNCTION FUNC(X,Y,I)  (actual parameter for I is REAL)
```

520 (PLDIM) -- Search for corresponding entries (parameters) in corresponding parameter lists which do not have compatible dimensions. Such constructions are highly error-prone. The model for comparison is always the formal (dummy) parameter list of a subprogram. The name of an external reference (a subprogram) is considered to be the first entry in its own parameter list. All messages appear in the primary listing.

Corresponding parameters do not have compatible dimensions if

1) The actual parameter is an array and the dummy parameter is a scalar.
2) The actual parameter is an element of an array and the dummy parameter is an entire array, except
   a) when both parameters have the same number of dimensions and
   b) the subscripts of the actual parameter are all ones, e.g.,

```
CALL SUB(A(1,1),N,M)
SUBROUTINE SUB(B,N,M)
DIMENSION B(N,M)
```

3) The actual parameter and the dummy parameter are both arrays, but they do not have the same number of dimensions.
4) The actual parameter and the dummy parameter are both arrays, they have the same number of dimensions, but the dimensions are not identical, except
   a) when the dummy array has dummy dimension, e.g.,

```
DIMENSION A(N)
```

   or

   b) when the dimensions of the dummy array are all ones, e.g.,

```
DIMENSION A(1,1)
```

Each message contains

1. The name of the parameter.
2. The number of subscripts the parameter has, from zero to n.
3. The parameter number. This indicates where the parameter appears in the parameter list.
4. If the parameter is an array, the names of the subscripts.
5. The name of the module which contains the corresponding parameter list.

521 (PLDIM) - Same as 520, except that the messages appear in the secondary listing and that each message contains, for both the formal parameter list and for the actual parameter list,

1. The first statement in the module.
2. The statement containing the parameter list.
3. The name of the parameter.
4. The number of subscripts the parameter has, from zero to n.
5. The parameter number. This indicates where the parameter appears in the parameter list.
6. If the parameter is an array, the names of the subscripts.

Example:
```
DIMENSION A(10),B(5),C(10),D(10,3)
CALL SUB (A,B(2),C,D)

SUBROUTINE SUB(X,B,C,D)
DIMENSION B(5),C(11),D(3,10)
```

600 (CYCALL) - Search for cyclic calling sequences. Such calling sequences are illegal. All messages appear in the display listing.

The message contains the names of the routines involved in the cyclic calling sequence. No source code statements are displayed.

Example:
```
SUBROUTINE A
CALL B

SUBROUTINE B
CALL C

SUBROUTINE C
CALL A
```

Query Peculiarities

130 DATVAR - Flag DATA statements not in BLOCK DATA which contain COMMON Block variables.

If a DATA statement refers to a variable which appears in an EQUIVALENCE list, then all members of the EQUIVALENCE list are examined.

140 FUNPAR - Flag function dummy parameters which are assigned value within the function itself.

If a function dummy parameter appears in an EQUIVALENCE list, then all members of the EQUIVALENCE list are examined.

Program boundaries are not crossed. The use of a dummy parameter in an actual parameter list is ignored, e.g.,

```
FUNCTION FUN (X,Y)
      .
      .
      .
CALL  SUB (X,Z)
```

The use of X in the actual parameter list SUB is ignored while processing this query.

160 REDLOP - Flag DO Loop control variables which are assigned values within the loop itself.

If a control variable appears in an EQUIVALENCE list, then all members of the EQUIVALENCE list are examined.

170 & 171 DOTERM - Flag DO loop index variables which are used after the loop has terminated under normal conditions.

If a DO loop index variable appears in an EQUIVALENCE list, then all members of the EQUIVALENCE list are examined. If there exists a path such that a DO loop index variable is used after the path terminated normally, a warning message is printed.

Example 1:

```
5            DO  100 I = 1, K
6                IF  (I.GT.3)  GO TO  110
7      100   CONTINUE
8      110   J = I
```

In Query 171, the path(s) leading to a DO loop index variable used after the loop terminated normally are printed, as well as the beginning and end of the DO loop. The above example would produce in a secondary report.

```
5            DO  100 I = 1, K
7      100   CONTINUE
8      110   J = I
```

A certain amount of path tracing is performed for this query. If a DO loop index variable or a variable EQUIVALENCED to it appears in an actual parameter list outside the DO loop, then it is assumed that the variable is assigned a value by that external reference. This assumption is made because the query does not cross program boundaries.

Example 2:

```
      DO  100 I = 1, 10
          .
          .
          .
100   CONTINUE
      K = I
```

Example 3:

```
      DO  100 I = 1, 10
          .
          .
          .
100   CONTINUE
      CALL  SUB(I)
      K = I
```

Example 2 receives a warning flag, Example 3 does not.

180 ASNUSE - Flag local variables which are assigned values but never used.

If a local variable appears in an EQUIVALENCE list, then all members of the EQUIVALENCE list are examined.

This query does not differenctiate between an array and an element of the array. If one element of the array is used, then it is assumed that every element of the array is used.

190 & 191 UNINT - Flag uninitialized local variables.

If a local variable appears in an EQUIVALENCE statement, then all members of the EQUIVALENCE list are examined. If there exists a path such that a local variable is uninitialized, a warning message is printed.

Example 1:

```
      5          IF (J.LE.0) GO TO 10
      6          L = 5
      7          K = 1
      8          GO TO 20
      9     10   L = 7
     10     20   J = K+L
```

In Query 191, the path(s) leading to the use of an uninitialized variable are printed. The above example would produce in a secondary report

```
      5          IF (J.LE.0) GO TO 10
      9     10   L = 7
     10     20   J = K+L
```

This query does not differentiate between an array and an element of the array. If one element of the array is assigned a value, then it is assumed that every element of the array is assigned a value.

Example 2:

```
      SUBROUTINE  SUB
      DIMENSION  A(10)
      B = A(1)
```

Example 3:

```
      SUBROUTINE  SUB
      DIMENSION  A(10)
      A(2) = 1.1
      B = A(1)
```

Example 2 receives a warning flag, Example 3 does not.

While searching for uninitialized variables, a certain amount of path tracing is performed. If a variable or a

variable EQUIVALENCED to it, appears in an actual parameter list, then it is assumed that the variable is assigned a value by that external reference. This assumption is made because the query does not cross program boundaries.

Example 4:

```
      SUBROUTINE SUB
      K = I
```

Example 5:

```
      SUBROUTINE  SUB
      CALL  SUBRT(I)
      K = I
```

Example 4 receives a warning flag, Example 5 does not.

COMMON Block Misalignment:

The queries dealing with COMMON Block misalignment share a number of peculiarities and limitations.

To begin with, queries which compare individual COMMON Block entries do not produce any warning messages if an entry in one COMMON Block declaration does not have a corresponding entry in another declaration. For example,

```
SUBROUTINE  SUB1            SUBROUTINE  SUB2
COMMON/BLOCK/ A,B,C         COMMON/BLOCK/ A,B
```

No warning messages which concern type, dimensionality, individual entry size, or name mismatch are printed for variable C. However, warning messages are produced by those queries concerned with the total size of the COMMON Block and the number of entries in the COMMON Block.

Secondly, the COMMON Block misalignment checks use the first appearance of a COMMON Block as the model for comparison of all occurrences of the COMMON Block. Since modules are stored in alphabetical order, a COMMON Block declaration residing in a module which is at the beginning of the alphabetical listing is always used as the model. The primary listing contains the name of the module which contains the COMMON Block declaration being compared against.

One limitation is that after the first mismatch in a COMMON Block declaration is found by a query, the query halts processing on that COMMON Block declaration. For example,

```
SUBROUTINE  SUB1            SUBROUTINE  SUB2
COMMON/COM/ A,B,C           COMMON/COM/ I,J,K
```

Type mismatch warning messages appear only to A and I, and not for B, C, J, or K.

Another limitation is that only the statement containing the first appearance of a given COMMON Block in a module is printed in the secondary listing. For example,

```
SUBROUTINE  SUB
COMMON/COM/ A,B,C
COMMON/COM/ D,E
```

In a secondary listing, only

```
COMMON/COM/ A,B,C
```

appears.

400 & 401 CBNENT - Flag corresponding COMMON Block declarations which do not have the same number of entries.

```
SUBROUTINE  SUB1            SUBROUTINE  SUB2
COMMON/COM1/X,Y,Z           COMMON/COM1/X,Y
COMMON/COM2/ A(2)           COMMON/COM2/ C,D
```

Warning messages are printed for both /COM1/ and /COM2/

410 & 411 CBTYPE - Flag corresponding entries in COMMON Block declarations which do not have the same type.

```
SUBROUTINE  SUB1            SUBROUTINE  SUB2
COMMON/COM1/ A,B,C          COMMON/COM1/ A,B,I
```

Warning messages are printed for C and I in /COM1/

420 & 421 CBDIM - Flag corresponding entries in COMMON Block declarations which do not have identical dimensions. Scalars are defined as having zero dimensions.

```
SUBROUTINE  SUB1          SUBROUTINE  SUB2
COMMON/COM1/ A            COMMON/COM1/ A(1)
COMMON/COM2/ B(2,3)       COMMON/COM2/ B(3,2)
```

Warning messages are printed for A in /COM1/ and for B in /COM2/

440 & 441 CBNAME - Flag corresponding entries in COMMON Block declarations which do not have identical names.

```
SUBROUTINE  SUB1          SUBROUTINE  SUB2
COMMON/COM1/ A,B,C        COMMON/COM1/ A,B,X
COMMON/COM2/ D(2)         COMMON/COM2/ D,E
```

Name mismatch messages are printed for C and X of /COM1/ and for D and E of /COM2/

450 & 451 CBINDS - Flag corresponding entries in COMMON Block declarations which do not have the same individual size. For the following example, assume integers are one word long and reals are two words long.

```
SUBROUTINE  SUB1              SUBROUTINE  SUB2
                              INTEGER B
COMMON/COM1/ A,B,C            COMMON/COM1/ A,B,C
COMMON/COM2/ D(2,3), E(3)     COMMON/COM2/ D(3,2), E(2)
```

Individual size mismatch warning messages are printed for B of /COM1/ and E of /COM2/

460 & 461 CBTOTS - Flag corresponding COMMON Blocks which do not have the same total size.

SUBROUTINE SUB1	SUBROUTINE SUB2
COMMON/COM1/ A,B,C	COMMON/COM1/ A,B,C,D
COMMON/COM2/ X,Y(2,2)	COMMON/COM2/ Q,R,S,T,U

Warning messages are printed for /COM1/

Parameter List Misalignment:

The queries dealing with parameter list misalignment share a number of peculiarities and limitations.

To begin with, queries which compare individual parameters do not produce any warning messages if two corresponding parameter lists do not have the same number of parameters. For example,

```
CALL  $UB(X,Y,Z)
  .
  .
  .
SUBROUTINE  SUB(A,B)
```

No warning messages concerning type or dimensionality mismatch are printed for Z. However, a warning message is printed by the query which compares the number of parameters appearing in corresponding parameter lists.

Secondly, the parameter list misalignment checks use formal (dummy) parameter lists as the models for comparison.

Thirdly, the name of the parameter list is processed as the 0^{th} parameter in the parameter list. For example, in

```
FUNCTION  FUN(X,Y)
```

FUN is the 0^{th} parameter, X the 1^{st}, and Y the 2^{nd}.

Finally, all parameters in a parameter list are examined. If a parameter is a subexpression, then the parameter is assumed to have the name *SUBEXPR.

500 & 501 PLNENT - Flag corresponding parameter lists which do not have the same number of parameters.

```
CALL  SUB1                    CAL  SUB2 (X+Y)
SUBROUTINE  SUB1(K)           SUBROUTINE  SUB2(Z)
```

Warning messages are printed for SUB1.

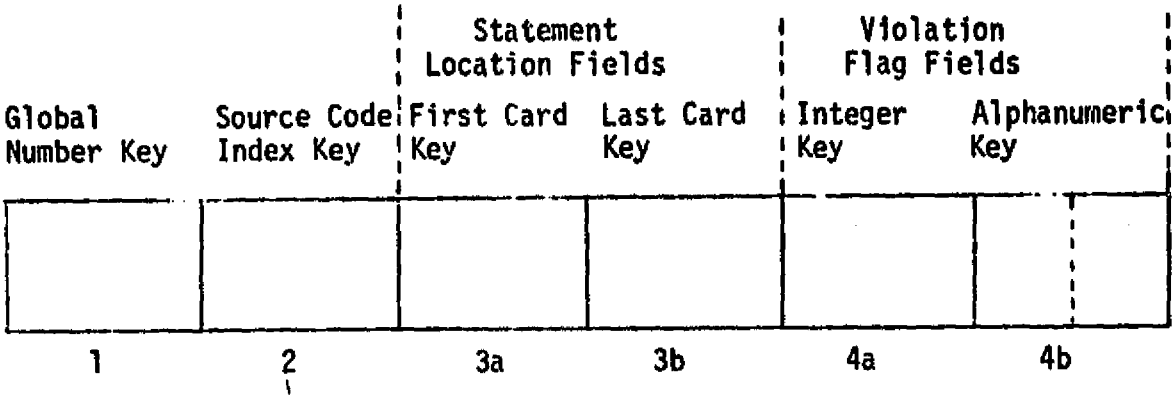
510 & 511 PLTYPE - Flag corresponding parameters which do not have the same type.

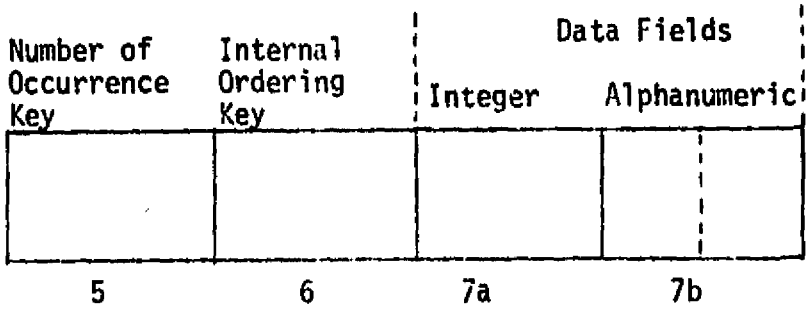
```
IMPLICIT REAL (A-Z)
K = FUN (X,Y)
    .
    .
    .
INTEGER FUNCTION FUN (X,Y)
INTEGER Y
```

Warning messages are printed for parameters FUN and Y.

520 & 521 PLDIM - Flag corresponding parameters which do not have compatible dimensions. See Queries for a more detailed discussion.

Flag File (Sort File) Format





There are seven sort fields and three data fields. All are integer fields, except the alphanumeric Flag Field and the alphanumeric Data Field. Any field, alphanumeric or integer, which does not contain any information contains a zero. The output format is

FORMAT(5(2X,I5), 2X, 2A4, 3(2X,I5), 2X, 2A4)

1. Global Number Key - This sort key specifies whether the violation is to appear in the primary listing or in the secondary or display listings. A one in this key indicates that the violation is to

appear in the primary listing. An integer >1 indicates that the violation is to appear in the secondary or display listing.

2. Source Code Index Key - This sort key specifies the location of the beginning of the source code for the module in which the violation occurs. If the violation does not occur in a specific module, as in cyclic calls, then this key is zero.

3. Statement Location Fields - This area consists of two sort fields, the First Card Key and the Last Card Key.
   a. First Card Key - This sort key specifies the first card of the statement in which the violation occurs. If the violation does not occur in a specific card, as in cyclic calls, then this key contains a zero.
   b. Last Card Key - This sort key specifies the last card of the statement in which the violation occurs. If the violation does not occur in a specific card, as in cyclic calls, then this key contains a zero.

4. Violation Flag Fields - This area consists of two fields, an integer sort field and an alphanumeric non-sort field.
   a. Integer Violation Flag Key - This sort key specifies which violation has occurred. Each type of violation has its own integer code. If a violation is to appear within the primary listing or the secondary listing, this key specifies which.
   b. Alphanumeric Violation Code - This field contains the alphanumeric name of the violation.

5. Number of Occurrence Key - This sort key keeps track of the order in which violations of the same class occurred.

   If the query searches for local violations, then this sort key is set to zero each time a different module is examined, and incremented each time a violation occurs.

   If the query searches for global violations not involving path tracing (Parameter List Alignment and COMMON Block Alignment), then this sort key is set to zero when the query is invoked, and incremented each time a violation occurs.

   If the query searches for global violations using path tracing (Cyclic Call Search), this sort key is set to zero each time the query starts at a new path beginning.

6. Internal Ordering Key - This sort key specifies the internal ordering of the data concerning a violation. Since a violation may involve a great deal of information to be passed on to the user, this information must be placed in some order. For example, in a dimensional mismatch, the violation information would include (in the Data Fields) the name of the variable in violation, how many dimensions it has, and what those dimensions are. These pieces of data must retain their proper order for the output to be intelligible.

7. Data Fields - This area consists of two fields, an integer data field and an alphanumeric data field.

   a. Integer Data Field - This field contains integer data that describes or pinpoints the violation, such as the size of a COMMON Block.

b. Alphanumeric Data Field - This field contains alphanumeric data that describes or pinpoints the violation, such as the name of a variable.

Flag File Peculiarities

1. Unless otherwise noted, all warning messages appear in the primary listing.
2. Unless otherwise noted, all warning messages are directly attached to the FORTRAN statements the messages reference.
3. In the Global Number column, N is assigned values during execution such that N>1. For a further discussion on the contents of this column, see the discussion on the Global Number COMMON Block, /GLO/.
4. In the Internal Ordering column, there are entries of the form '(0) 1'. This refers to the problem of variables being equivalenced to other names. If the variable is not equivalenced to another name, or if the equivalenced name is not part of the violation, then the internal ordering column contains a zero and no equivalenced name appears in the flag file. On the other hand, if a variable and name equivalenced to it are part of a violation, then the internal ordering column will contain a '1' and '2' respectively.
5. In the Integer column of the DATA field,
   a. 'entry number' indicates that a variable is the n^{th} COMMON Block variable in a COMMON Block declaration.
   b. 'parameter number' indicates that a parameter is the n^{th} parameter in a parameter list.
6. In dimensionality misalignment searches, a variable may have anywhere from zero to n dimensions.
7. For Parameter List Alignment, the model parameter list is always the formal (dummy) parameter list, while the comparison parameter list is always the actual parameter list.

8. In the Flag Integer Column, a '1' indicates that a statement is to be printed by the Report Generator, but that no message is to be printed with it.

FORTRAN FRONT END FLAGS	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
Statement not processed	1	012	NO PROC	0			Not used in processing
Unrecognized Statement	1	013	UNRECOG	0			Statement not in FFE processing set
Statement Truncated	1	022	STM TRUC	1 2 3 ⋮ N		Display of form at truncation point 1 2 3 4 ⋮	Statement process terminated before statement completed
Statement Aborted	1	023	SABORT	1 ⋮ N		Display of form at stop position	Statement process halted for syntax problem
Branch List Truncated	1	042	BR TRUC	0			Branch list of statement truncated due to overload
Node Table Overflow	1	043	NOD FULL	0			Node Table full. No more statements added to tables for module
Predecessor Table Overflow	1	044	PRE FULL	0			No predecessors added to statements which follow

FORTRAN FRONT END FLAGS	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Successor Table Overflow	1	045	SUC FULL	0			No successors for nodes that follow
Use Table Overflow	1	046	USE FULL	0			Uses for statements which follow not recorded
Symbol Table Overflow	1	052	NO SYM	0		Symbol (8 chars.)	Symbol not added to symbol table
Symbol Truncated	1	053	SYM TRUC	1 2 . . . N		Symbol string of truncated form	Truncated form used internally
Header Card not Found	1	062	NO HEAD	0			Header card not found after last module end
End Statement Missing	1	063	NO END	0			No end card on module
Unrecognized Symbol	1	082	SYM UREC	0		Symbol	Unknown character or context of use
Undefined Statement Label	1	083	NO DEFN	0		Label	Branch label
Parenthesis too Deep	1	084	PARENS	0			Parenthesis too deep in nested v ( ) forms

FORTRAN FRONT END FLAGS	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Unable to Locate Desired Symbol	1	085	MISSING	0		Symbol	Looking for symbol but cannot find
Unknown Form	1	086	FORM	1 2 3 . . . N		Display of unknown form	This form of construction not recognized
Card Exhausted while Trying to Complete an Element	1	087	TOO SOON				Hollerith constants

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Search for ANSI Standards Function Names used not as ANSI Standards Functions	1	110	ANSIST	0		ANSI Standards Name	
Search for FORTRAN Reserved Words used as names	1	120	RESWRD	0		FORTRAN Reserved Word	
Search for DATA statements containing COMMON Block variables not in BLOCK DATA	1	130	DATVAR	(0) 1		Name of Variable	
	1	130	DATVAR	2		Equivalenced Name	
Search for function dummy parameters assigned value within the function itself	1	140	FUNPAR	(0) 1		Name of dummy parameter	
	1	140	FUNPAR	2		equivalenced name	
Search for multiple branching statements which do not branch to the statement immediately following	1	150	MULBRA	0			
Search for the redefinition of DO Loop control variable within the loop itself	1	160	REDLOP	(0) 1		Name of control variable	
	1	160	REDLOP	2		Equivalenced Name	

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Search for DO Loop index variable used after the loop has terminated normally							
Warning messages to appear in primary listing	1	170	DOTERM	(0) 1		Name of index variable	
	1	170	DOTERM	2		Equivalenced name	
Warning messages to appear in secondary listings	1	1	DOTERM	1			1st statement in module
	1	1	DOTERM	2			statement containing beginning of DO Loop
	1	1	DOTERM	3			statement containing end of DO Loop
	1	1	DOTERM	4		Name of index variable	1st statement in path
	⋮	⋮	⋮	⋮		⋮	⋮
	1	1	DOTERM	n+2		Name of index variable	n-1st statement in path
	N	171	DOTERM	n+3		Name of index variable	nth statement in path
	N	171	DOTERM	n+4		Equivalenced name	
Search for local variables assigned values but never used	1	180	ASNUSE	0		Name of variable	

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Search for uninitialized local variable							
Warning messages to appear in primary listing	1	190	UNINT	0		Name of variable	
Warning messages to appear in secondary listing	1	1	UNINT	1			1st statement in module
	1	1	UNINT	2		Name of variable	1st statement in path
	⋮	⋮	⋮	⋮		⋮	⋮
	1	1	UNINT	n+1		Name of variable	n-1st statement in path
	N	191	UNINT	n+1		Name of variable	nth statement in path

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding COMMON Block declarations have the same number of entries							
	1	400	CBNENT	1	Number of entries	Name of COMMON Block	
	1	400	CBNENT	2		Name of the other module	
Warning messages to appear in secondary listings	N	1	CBNENT	1	1 (model indicator)		1st statement in model module
	N	401	CBNENT	2		Name of COMMON Block	
	N	401	CBNENT	3	Number of entries		
	N	1	CBNENT	4	2 (comparison indicator)		1st statement in comparison module
	N	401	CBNENT	5		Name of COMMON Block	
	N	401	CBNENT	6	Number of entries		

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding entries in corresponding COMMON Block declarations have the same type							
Warning messages to appear in primary listing	1	410	CBTYPE	1	Entry's data type	Name of COMMON Block	
	1	410	CBTYPE	2	Entry number	Name of entry	
	1	410	CBTYPE	3		Name of other module	
Warning messages to appear in secondary listing	N	1	CBTYPE	1	1 (model indicator)		1st statement in model module
	N	411	CBTYPE	2	Entry's data type	Name of COMMON Block	
	N	411	CBTYPE	3	Entry Number	Name of entry	
	N	1	CBTYPE	4	2 (comparison indicator)		1st statement in comparison module
	N	411	CBTYPE	5	Entry's data type	Name of COMMON Block	
	N	411	CBTYPE	6	Entry number	Name of entry	

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding entries in corresponding COMMON Block declarations have matching dimensions							
Warning messages to appear in primary listing	1	420	CBDIM	1	Number of dimensions	Name of COMMON Block	
	1	420	CBDIM	2	Entry Number	Name of entry	
	1	420	CBDIM	3	1st dimension	Name of other module	1st dimension does not exist if entry is a scalar
	⋮	⋮	⋮	⋮	⋮		
	1	420	CBDIM	n+2	nth dimension		
Warning messages to appear in secondary listing	N	1	CBDIM	1	1 (model indicator)		1st statement in model module
	N	421	CBDIM	2	Number of dimensions	Name of COMMON Block	
	N	421	CBDIM	3	Entry number	Name of entry	
	N	421	CBDIM	4	1st dimension		
	⋮	⋮	⋮	⋮	⋮		
(continued)	N	421	CBDIM	n+3	nth dimension		

Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
(continued from preceding page)	N	1	CBDIM	n+4	2 (comparison indicator)		1st statement in comparison module
	N	421	CBDIM	n+5	Number of dimensions	Name of COMMON Block	
	N	421	CBDIM	n+6	Entry number	Name of entry	
	N	421	CBDIM	n+7	1st dimension		
	⋮	⋮	⋮	⋮	⋮		
	N	421	CBDIM	n+m+6	mth dimension		
Determine if a COMMON Block appears in only one module	1	430	CBONE	0		Name of COMMON Block	

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding entries in corresponding COMMON Block declarations have identical names							
Warning messages to appear in primary listing	1	440	CBNAME	1		Name of COMMON Block	
	1	440	CBNAME	2	Entry Number	Name of entry	
	1	440	CBNAME	3		Name of other module	
Warning messages to appear in secondary listing	N	1	CBNAME	1	1 (model indicator)		1st statement in model module
	N	441	CBNAME	2		Name of COMMON Block	
	N	441	CBNAME	3	Entry number	Name of entry	
	N	1	CBNAME	4	2 (comparison indicator)		1st statement in comparison module
	N	441	CBNAME	5		Name of COMMON Block	
	N	441	CBNAME	6	Entry number	Name of entry	

Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
Determine if corresponding entries in corresponding COMMON Block declarations have the same individual size							
Warning messages to appear in primary listing	1	450	CBINDS	1	Size of entry	Name of COMMON Block	
	1	450	CBINDS	2	Entry Number	Name of entry	
	1	450	CBINDS	3		Name of other module	
Warning messages to appear secondary listing	N	1	CBINDS	1	1 (model indicator)		1st statement in model module
	N	451	CBINDS	2	Size of entry	Name of COMMON Block	
	N	451	CBINDS	3	Entry number	Name of entry	
	N	1	CBINDS	4	2 (comparison indicator)		1st statement in comparison module
	N	451	CBINDS	5	Size of entry	Name of COMMON Block	
	N	451	CBINDS	6	Entry Number	Name of entry	

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding COMMON Blocks have the same total size							
Warning messages to appear in primary listing	1	460	CBTOTS	1	Total size	Name of COMMON Block	
	1	460	CBTOTS	2		Name of other module	
Warning messages to appear in secondary listing	N	1	CBTOTS	1	1 (model indicator)		1st statement in model module
	N	461	CBTOTS	2		Name of COMMON Block	
	N	461	CBTOTS	3	Total size		
	N	1	CBTOTS	4	2 (comparison indicator)		1st statement in comparison module
	N	461	CBTOTS	5	Total size	Name of COMMON Block	
	N	461	CBTOTS	6	Total size		

Query	Global Number	FLAG Integer	FLAG Alpha-numeric	Internal Ordering	DATA Integer	DATA Alpha-numeric	Comments
Determine if corresponding parameter lists have the same number of parameters							
Warning messages to appear in primary listing	1	500	PLNENT	1	Number of parameters	Name of parameter list	
	1	500	PLNENT	2		Name of other module	
Warning messages to appear in secondary listing	N	1	PLNENT	1	1 (model indicator)		1st statement in model module
	N	501	PLNENT	2		Name of parameter list	
	N	501	PLNENT	3	Number of parameters		
	N	1	PLNENT	4	2 (comparison indicator)		1st statement comparison module
	N	501	PLNENT	5		Name of parameter list	
	N	501	PLNENT	6	Number of parameters		

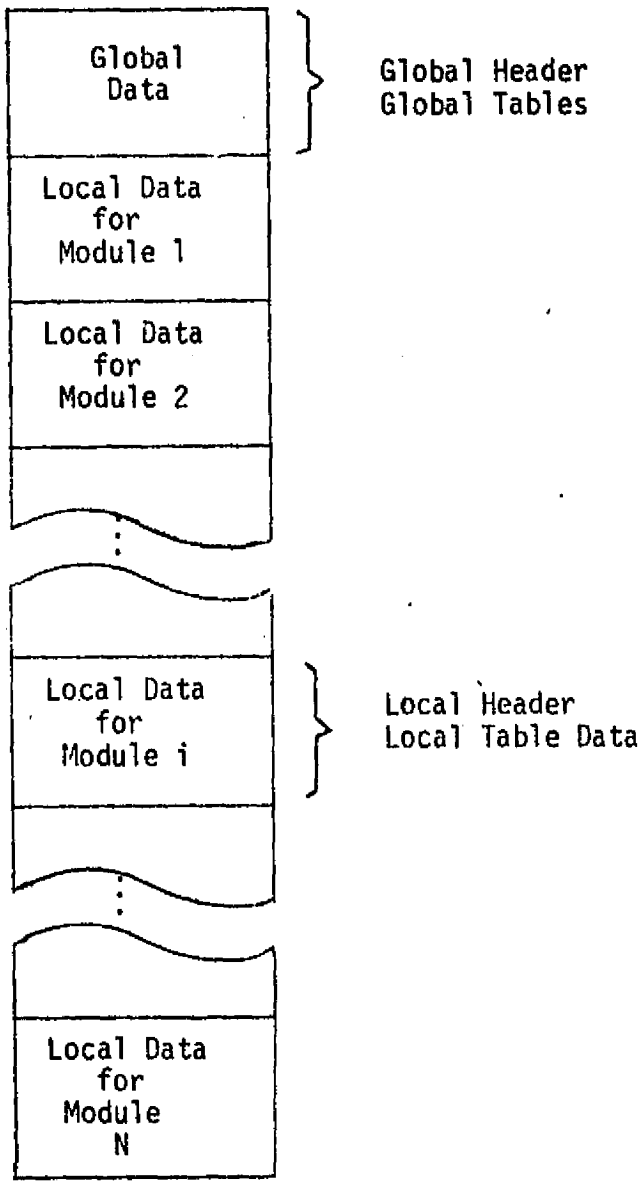
Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
Determine if corresponding parameters in corresponding parameter lists have the same type							
Warning messages to appear in primary listing	1	510	PLTYPE	1	Parameter's type	Number of para-list	
	1	510	PLTYPE	2	Parameter Number	Name of parameter	
	1	510	PLTYPE	3		Name of other module	
Warning messages to appear in secondary listing	N	1	PLTYPE	1	1 (model indicator)		1st statement in model module
	N	511	PLTYPE	2	Parameter's type	Name of parameter list	
	N	511	PLTYPE	3	Parameter number	Name of parameter	
	N	1	PLTYPE	4	2 (comparison indicator)		1st statement in comparison module
	N	511	PLTYPE	5	Parameter's type	Name of parameter list	
	N	511	PLTYPE	6	Parameter number	Name of parameter	

Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
Determine if corresponding parameters in corresponding parameter lists have compatible dimensions							
Warning messages to appear in primary listing	1	520	PLDIM	1	Number of subscripts	Name of parameter list	
	1	520	PLDIM	2	Parameter Number	Name of parameter	
	1	520	PLDIM	3		Name of other module	
	1	520	PLDIM	4		Name of 1st subscript	
	⋮	⋮	⋮	⋮		⋮	
	1	520	PLDIM	n+3		Name of nth subscript	

Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
(continued)							
Warning messages to appear in secondary listing	N	1	PLDIM	1	1 (model indicator)		1st statement in model module
	N	521	PLDIM	2	Number of subscripts	Name of parameter list	
	N	521	PLDIM	3	Parameter number	Name of parameter	
	N	521	PLDIM	4		Name of 1st subscript	
	⋮	⋮	⋮	⋮		⋮	
	N	521	PLDIM	n+3		Name of nth subscript	
	N	1	PLDIM	n+4	2 (comparison indicator)		1st statement in comparison module
	N	521	PLDIM	n+5	Number of subscripts	Name of parameter list	
	N	521	PLDIM	n+6	Parameter number	Name of parameter	
	N	521	PLDIM	n+7		Name of 1st subscript	
	⋮	⋮	⋮	⋮		⋮	
	N	521	PLDIM	n+m+6		Name of mth subscript	

Query	FLAG				DATA		Comments
	Global Number	Integer	Alpha-numeric	Internal Ordering	Integer	Alpha-numeric	
Search for cyclic calling sequences warning messages to appear in display listing	N	600	CYCALL	1		Name of calling routine	1st member of cyclic sequence
	N	600	CYCALL	2		Name of calling routine	2nd member of cyclic sequence
	⋮	⋮	⋮	⋮		⋮	⋮
	N	600	CYCALL	n		Name of calling routine	nth member of cyclic sequence (same as beginning of sequence)

Table File Structure



Global Table Allocation

Associated Common Block		Starting Record	Length (words)
/GHD/	GLOBAL HEADER	1	28
/DIR/	MODULE DIRECTORY	2	800
/SH/	SYSTEM HIERARCHY TABLE	10	400
/SHD/	SYSTEM HIERARCHY TO DIRECTORY TABLE	14	400
/IS/	INVERSE SYSTEM HIERARCHY TABLE	18	400
/ISD/	INVERSE SYSTEM HIERARCHY TO DIRECTORY TABLE	22	400
/COM/	COMMON BLOCK NAME TABLE	26	300
/LIN/	LINK LIST FOR COMMON NAMES TABLE	29	1000

Local Table File Structure for a Module

Associated Common Block		Starting Record Number	Length in Words
/MHD/	LOCAL HEADER	N	6
/SYM/ SYMTAB	MAIN SYMBOL TABLE	N+1	2800
/SYM/ SYMOVR	SYMBOL OVERFLOW TABLE	N+28	200
/USE/	USE TABLE	N+30	4000
/NOD/	NODE TABLE	N+70	2800
/SUC/	SUCCESSOR TABLE	N+98	1000
/PRE/	PREDECESSOR TABLE	N+108	1000

where N determined from module number entry of the Directory.